# The Universal Serial Bus : How it Works and What it Does

By Geoff Knagge (9806135)

USB is a system for connecting a wide range of peripherals to a computer, including pointing devices, displays, and data storage and communications products. Although not a relatively new development in personal computing, it has only recently gained popularity due to increasing software support. This document discusses what the USB system does and how it is done. The technical detail covers the system's logical structure, rather than the electrical or software characteristics.

## About this document

· This document is found on-line at http://members.xoom.com/gogogeoff/uni/elec101.
· I can be contacted by email at geoff.knagge@studentmail.newcastle.edu.au
· Where new technical terms are introduced and defined in the text, they are printed in ***bold italics***

## Document Contents

## Appendices

Traditionally, computer hardware such as printers and mice were plugged into sockets on the back panel of a PC, with each connector being fairly specialised in its applications. For example, mice use one that is dedicated to transmitting serial data, while printers usually use a parallel data cable, and monitors have their own special plug. This system was quite suitable when it was implemented and has been used for many years.

However, recent technological developments have created problems for many users of PCs with this system. Today there is a huge range of peripheral equipment including scanners, digital cameras, specialised pointing devices, high speed modems, all of  which need their own connection to the PC. While the above mentioned parallel and serial sockets can indeed be used by many different devices, they cannot be shared by more than one device at once, and so we can quickly run out of space to attach this new equipment.

Then why not just add more places to plug  things into? This is quite possible, and has previously been the solution in many cases by installing more sockets, and even connecting devices internally to bypass the need for these plugs altogether. However, there is a practical limit to how much this can be done. To examine why, we need to investigate the structure of the PC.

Each of those back panel sockets is attached to a circuit board (a card) which, like the internal devices, in turn plugs into a slot on the main circuit board. The computer needs to be able to distinguish which of these cards information is coming from, and similarly each card needs to know when out going data is directed to it. This is done by using special numbers called IRQs ("Interrupt Request"), used for when the card wants to get something done, and I/O ("Input/Output") addresses, used for transferring data between the computer and cards. When the card has something for the software to do, it generates a signal on the IRQ line, and communicates with the software via the I/O addresses. The IRQs and I/O addresses are set in hardware on the card but, while these are usually adjustable, people have often found that when adding new hardware these numbers can clash with existing cards. The result is that neither card functions correctly.

Many people find these potential problems rather daunting and this reduces the attractiveness of PCs to the market. An innovation known as "Plug and Play" aims to sort out the problem of clashing numbers, but there are still only a finite amount of these numbers available, with some being reserved for certain purposes. As well as all of the technical problems, where are other nuisances such as excessive cabling, the fact that you cannot disconnect devices while the computer is on without risking damaging something, and the different types of plugs required for different types of computers such as the Apple Macintosh. Even in the absence of problems, that back of a computer can be a daunting place for some, who often fear damaging something.

This is where USB aims to simplify things by extending the trend of "user friendliness" to the hardware level. To the average computer user, it is a system where you can simply plug a device into any available socket and that device will instantly be available for use by the computer. Up to 127 devices can be connected, and since it is a high speed system supporting up to 12 Megabits per second, it can accommodate the needs of a wide variety of peripherals. Other advantages include the ability to safely disconnect and reconnect items without switching off the computer, and the ability to use a USB device on any computer supporting the USB system.

The Universal Serial Bus is a network of attachments connected to the host computer. These attachments come in two types known as Functions and Hubs. ***Functions*** are the peripherals such as mice, printers, etc. ***Hubs*** basically act like a double adapter does on a power-point, converting one socket, called a ***port***, into multiple ports. Hubs and functions are collectively called ***devices***.

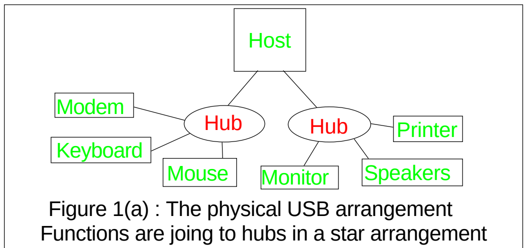

Figure 1(a) : The physical USB arrangement
Functions are joing to hubs in a star arrangement

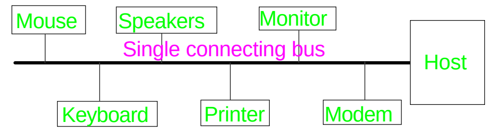

Figure 1(b) : How the USB system appears to functions

As far as the functions are concerned, hubs are furthermore like double adapters because although the entire system is physically in the star topology seen in Figure 1(a), logically the system acts like a bus topology. This means that signals appear to travel along a single set of wiring, called the ***bus***, to the host and is accessible by all functions, as illustrated in Figure 1(b). However, the host does keep track of the physical arrangements so that if a hub becomes disconnected, it is aware that all hubs and functions attached to it will consequently be disconnected too.

The host has a hub embedded in it called the ***root hub***, and in practical implementations hubs are usually combined with one or more functions, such as keyboards or monitors. These are called ***compound devices*** and act like a hub with the functions permanently connected, along with any additional ports. Hubs may be connected to other hubs in a tiered arrangement, but the bus topology still applies.

## 3. How Does Data Get to the Right Places?

Even with this type of arrangement, we still have the same problem as with the traditional PC layout. Each function has to know when a piece of data is meant for it, and the host needs to know where signals are coming from, so numbers are assigned to each component on the USB. However, rather than each using a fixed IRQ and I/O address, the USB system takes a different approach.

When a device is attached to the USB system, it gets assigned a number called its ***address***. The address is uniquely used by that device while it is connected and, unlike the traditional system, this number is likely to be different to the address given to that device the last time it was used. Each device also contains a number of ***endpoints***, which are a collection of sources and destinations for communications between the host and the device.

Endpoints operate in simplex mode, meaning that they are either an input or output, but not both. For example, a simplistic model of a keyboard (figure 2) could have a keypad as output endpoint number 1, and the LED key lock display as receiving endpoint 1. All USB devices have one of each of their 16 possible input and output endpoints reserved as "***zero endpoints***". These are used for the auto-detection and configuration of the device when it is connected, and are the only accessible endpoints until this occurs. In addition each endpoint sets, upon connection, its own set of characteristic requirements concerning its requirements when accessing the bus.

The combination of the address, endpoint number and direction are what is used by the host and software to determine along which pipe data is travelling. A ***pipe*** is simply a data path between an endpoint and the associated portion of the controlling software, such as between the Keyboard LEDs and the BIOS routine which determines what LEDs should be lit. A special pipe is defined to connect to the zero endpoints, and is called the ***Default Control Pipe***.

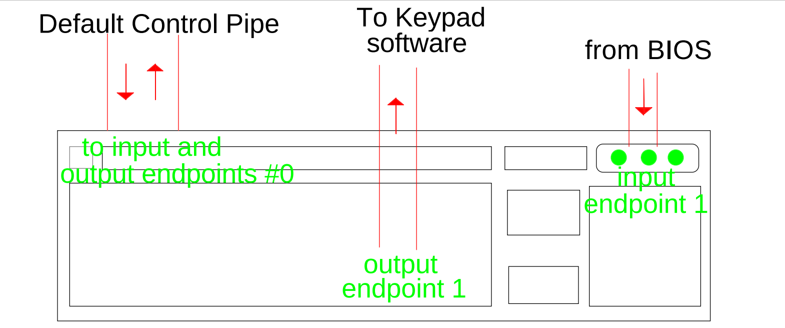

Figure 2 : Simple keyboard model illustrating endpoints and pipes

## 4. Types of data transfers

Before we can understand any further detail about the communications process within a USB network, we need to be aware of the types of data that it must cope with. To accommodate the different types of data that needs to travel across the USB, each pipe can be configured as one of four transfer types.

· ***Control Transfers*** : These differ from the other types in that they are intended for use in configuring, controlling, and checking the status of a USB device. A request is sent to the device from the host, and appropriate data transfers follow in the appropriate pipes. At some later stage, a status indicator is returned to the host. The pipe used for this type of data may be bidirectional, but uses the same numbered endpoint for each direction. In addition, a device only handles one control request at a time, with the host withholding outstanding requests until a status is returned on the one in progress. For example, the Default Control Pipe uses Control Transfers and accomplishes such tasks as initialising the device, and telling the host of the requirements of each of its endpoints. This type of pipe might also be used to control the operation of other pipes.

· ***Isochronous Transfers*** : These involve data whose accuracy is not critical and which is sent at a rate corresponding to some timing mechanism. For example, 44100KHz audio fits into this category since it doesn't have to be perfectly accurate and every 44100 samples indicates one second of audio. USB provides a special type of transfer for this data, giving it preference to guarantee a constant transmission rate with the required bandwidth. To ensure that the USB has enough time to handle the maximum data flow (1023 bytes) in each frame, a check is made during the initial configuration and the pipes will only be configured if this check is successful. This transfer method uses unidirectional pipes with no error handling procedures. Even though an error may be indicated in the status reply to a request, the pipe will not be halted and it is up to the software to decide what to do.

· ***Interrupt Transfers*** : These are used for small, infrequent transfers which require priority over other requests. As with Isochronous transfers, pipe configuration is granted on whether or not the system can handle the maximum packet size within the required time, with a further restriction that stops Interrupt and Isochronous Transfers from using more than 90% of any frame (discussed later) and stopping other transfers from occurring. The endpoint tells the host during configuration how often it should be polled for interrupt requests, and upon each polling returns a NAK signal if there is nothing to send. The use of this type of pipe is in some ways similar in purpose to the IRQ lines of the traditional peripheral system used in computers.

· ***Bulk Transfers*** : As the name suggests, the intended purpose is for transmitting large amounts of data. This type of transfer gets the lowest priority, so pipes using this method are only allowed to transmit when there is available bandwidth. This means that a heavily loaded USB may have relatively slow bulk transfers compared to one with is servicing few devices. This transfer type would be useful for sending data from devices like digital scanners.

## 5. Low Speed Devices

The description of USB devices so far has describe full speed USB peripherals, however there exists a special class of device known as low speed devices. These are simpler devices such as joysticks, and are the same as the full speed versions, except for the following restrictions:

- The maximum packet size for data transfers is 8 bytes
- They cannot use Isochronous nor Bulk Transfer pipes
- May only have 2 endpoints other than the zero endpoints

This simplification makes such devices easier, and thus cheaper, to design and implement.

## 6. How is data sent across the USB?

When the software requires data transfer to occur between itself and the USB, it sends a block of data called an ***I/O Request Packet (IRP)*** to the appropriate pipe, and the software is later notified when this request is completed successfully or terminated by error. Other than the presence of an IRP request, the pipe has no interaction with the USB. In the event of an error after three retry attempts, the IRP is cancelled and all further and outstanding IRPs to that pipe are ignored until the software responds to the error signal that is generated by sending an appropriate call to the USB. How exactly this is handled depends upon the type of device and the software.

As suggested by the name Universal Serial Bus, data transmission in the bus occurs in a serial form. Bytes of data are broken up and sent along the bus one bit at a time, with the least significant bit first as illustrated by figure 3.

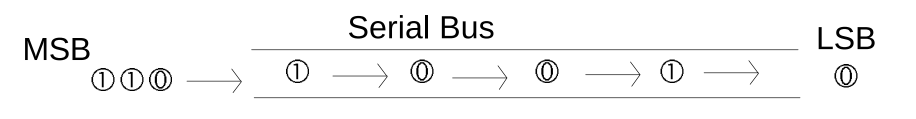

Figure 3 : Serial transmission of the binary number 11010010

The actual data is sent across the bus in packets. Each ***packet*** is a bundle of data along with information concerning the source, destination and length of the data, and also error detection information. Since each endpoint sets, during configuration, a limit to the size of the packet it can handle, an IRP may require several packets to be sent. Each of these packets should be the maximum possible size except for the final packet. The USB host has a built in mechanism so that the software can tell it when to expect full sized packets.

In the event that a less than maximum size packet is received earlier than expected, an error is assumed and the pipe is ***stalled*** with all IRPs being cancelled until the problem is dealt with by the controlling software. If an endpoint is busy, but no error has occurred, it responds with a special signal labelled NAK (Negative AcKnlowedge), which tells the other end of the pipe to wait a while. How these conditions are handled depends on the type of device and the software.

Each packet is made up of a set of components called ***fields*** including the following, summarised in figure 4 :

- An eight bit "***SYNC***" synchronisation field used by inputs to correct their timing for accepting data. Part of this field is a special symbol used to mark the start of a packet.
- The 8 bit Packet Identifier (***PID***) which uses 4 bits to determine the type, and hence format, of the packet data. The remaining 4 bits are a 1's complement of this, acting as check bits. Part of this field determines which of the four groups (token, data, handshake, and special) that the packet belongs to, and also specifies an input, output or setup instruction.
- An ***address field*** which gives the address of the function on the end of the pipe to be used
- The 4 bit ***endpoint field***, giving the appropriate endpoint which sends or receives the packet.
- A ***data field*** consisting of 0-1023 bytes

| Sync (8) | PID (8) | Address | Endpoint (4) | Data (0-1023 bytes) |
|---|---|---|---|---|

Figure 4 : A typical data packet. Numbers represent size of field in bits, unless otherwise indicated.

meaningful communication across the USB. Errors are detected by use of a Cyclic Redundancy Check (**CRC**) on all fields except the PID, which has its own checking mechanism. CRC is considered almost 100% accurate in error detection, and its process is described further in appendix A.

## 7. Examples of data transactions

A data transaction is simply a movement of data between the host and a connected device. The different types of possible transactions depend upon what transfer type the corresponding pipe is configured for.

### 7.1 Bulk Transfers

When the host wants to initiate a bulk transfer from a device, it sends a Token Packet specifying whether input or output is desired. If the host wants input, the function in turn sends either a packet of data or a non-ACK handshake. If the host is sending a request to transmit, it will immediately send the data packet and wait for an appropriate handshake to be returned by the recipient.

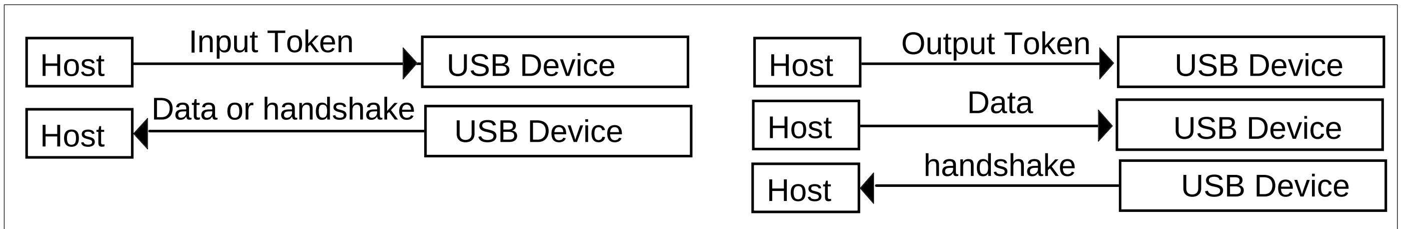

Figure 5 : Communcation sequence for input and output bulk transactions

### 7.2 Control Transactions

This is initiated by the host via an appropriately coded token packet with information regarding the control command being issued. The recipient either replies with an ACK handshake, or ignores the token totally. This type of transfer also contains an optional unidirectional data transfer between the host and the device. Lastly, a status is returned to whatever end of the pipe was last to transmit.

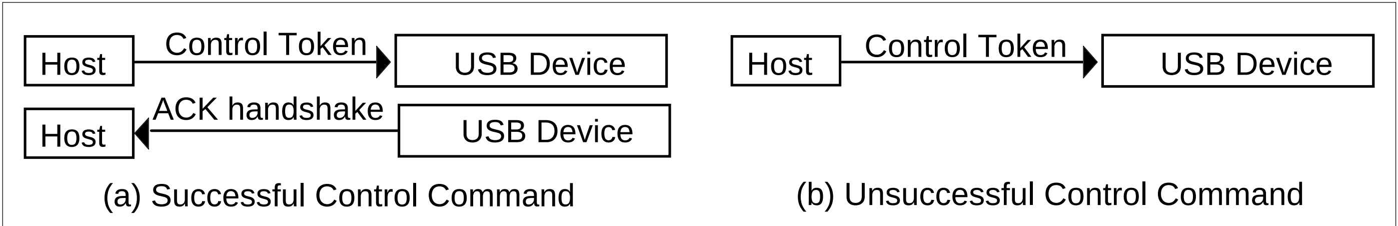

Figure 6 : Communcation sequence for Control transactions

### 7.3 Interrupt Transactions

The function is queried by the host via a token packet, and returns data if it has any interrupt related information to transmit, or a NAK handshake if it doesn't.

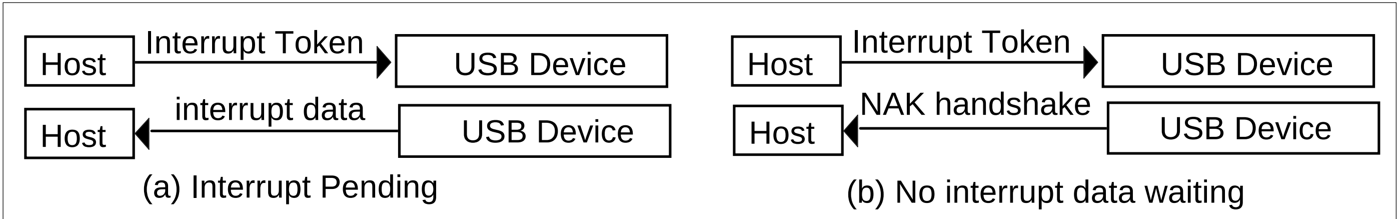

Figure 7 : Communcation sequence for Interrupt transactions

### 7.4 Isochronous Transactions

These are also begun with an appropriate token packet being sent to the function, and then data is transmitted as required. However, due to the assumed error-tolerance of isochronous data, no handshake packets are used.

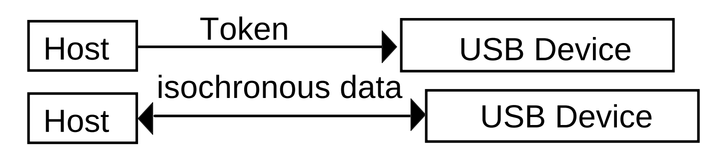

Figure 8 : Communcation sequence for Isochronous transactions

To the consumer, a hub simply expands one USB port into multiple ports. However, to manage this the hub must by able to be able to manage communications between itself and other devices, ensuring that only one transmission occurs on the bus at one time. Communications flows through the hub can be either from the host to a device, or vice versa.

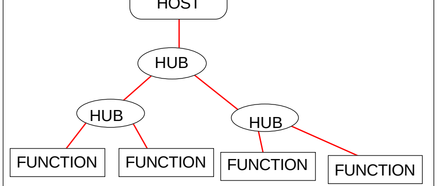

Figure 9 : Data flows from host to function
(red lines represent flow of data)

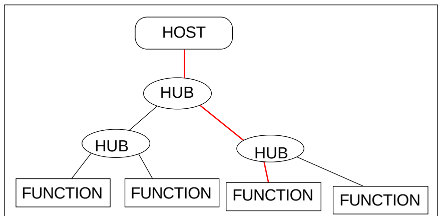

Figure 10 : Data flow from function to host
(red lines represent flow of data)

A hub goes into an idle state whenever there are no communications requests coming into it. When it detects data coming to it from wherever it connects to the USB, it immediately establishes a communications connection between itself and all devices connected to its ports, relaying that data as in figure 9. If a connected device has data to send to the host, the hub temporarily switches off all communications with other devices and relays the data to the point where it connects to the USB. This is then repeated by all further hubs (figure 10) until the data finally reaches the host.

The hub is also responsible for monitoring what devices are connected to its ports. When a connection or disconnection occurs, the host is notified by use of the status reply to a token packet. The host then sends another packet to the hub to inquire exactly what changes occurred. If it was a connection, a reset signal is sent to the appropriate port, an address is assigned to the device by the host, and a configuration sequence occurs via the default control pipe of that device. If a device is detached, the host updates its information to remove all devices on the disconnected chain from its records.

## 9. The Role of Functions

The whole point of the USB system is to define a communications protocol so that peripherals may communicate with their host computer. This is done via the various transaction types previously described, but the important concept in the USB system is that no functions may transmit as soon as they are ready to do so. They must wait to be queried, and then send the appropriate request via a reply or interrupt transfer.

As previously described, functions view the entire USB as a bus network of peripherals, oblivious to the role of hosts. They act no differently if they are connected 3 hubs down a chain than if they were directly connected to the root hub.

## 10. In Conclusion

The Universal Serial Bus offers a realistic alternative to the existing problem of configuring new peripherals for use with personal computers. The main advantage is the ease of use for the consumer, who simply has to plug the device into any available port for the device to immediately become available to appropriate software. In addition, USB devices are equally suited to any computer platform supporting the protocol, unlike current differences between some devices for the PC and Apple Macintosh. The option for devices to use a low speed mode allows cheaper peripherals such as joysticks to be designed for the system without the additional complexity. The only major disadvantage for the consumer is likely to be the cost of upgrading their computer in order to use the system. USB capable computers still support the traditional system too, so there is no problem with the compatibility of older hardware with such a computer.

The barriers which have slowed USB's growth in popularity have been mainly due to the unavailability of supporting software and hardware. The software problem is currently being overcome with USB support being included in the latest version of both Apple's and Microsoft's operating systems. Thus, there is now incentive for people to consider buying USB hardware, and hence for manufacturers to build it.

## Appendix A.  Cyclic Redundancy Checks

The Cyclic Redundancy Check, abbreviated CRC, is an error detection mechanism which is considered extremely accurate. Unlike simpler methods like parity checks and checksums, it is unlikely that errors within a transmission will cancel out and fool the checking circuitry.

In effect, CRC treats a stream of data bits as a large binary number, divides it by a prime, and transmits the remainder. However, this is hard to implement simply in circuitry, so the following method is used to obtain an equivalent result :

- The area of memory used for this calculation is filled with binary ones
- Each time a data bit is received, it is XORed with the high order bit
- The memory contents are shifted left one bit (multiply by 2), and the low order bit set to 0
- If the XOR produced a 1, then the register is XORed with a special polynomial known to both the receiver and transmitter
- After the last bit is sent, the CRC is inverted and sent for checking. If both transmitter and receiver do not get the same result, an error has occurred during transmission

## Appendix B. Types of Packets

To cope with the various communications that must occur to establish a data flow, there needs to be a variety of types of packets, and these are as follows:

- ***Token Packets*** : These are used to query the device and are issued by the host. They consist of  PID, address and endpoint fields, along with a 5 bit CRC check.
- ***Start-of-Frame Packets*** : The USB host controls the processing of data in 1ms units called frames. During each frame, it examines what requests are outstanding and allocates each pipe bandwidth depending on its requirements and type of transfer that it uses. Each frame is marked by a Start of Frame (SOF) packet, consisting of an appropriate PID, an 11-bit counter and a 5 bit CRC. The counter is incremented once per frame, allowing devices to determine whether they missed a frame due to an error and adjust their timing appropriately.
- ***Data Packets :*** Consists of all of the above field types, and is protected by a 16 bit CRC
- ***Handshake Packets*** : These are used for returning the status of data transfers and consist only of a PID. These come in three types and are named ACK (ACKnowledgement of receipt with no errors), NAK (the function is not ready to communicate data) and STALL (function is busy or some other error occurred)

## Appendix C. References

- My main source of information was the official USB 1.1 Specification, available at http://www.usb.org
- Some of the background information was from previous knowledge gained from various articles in the Electronics Australia and Australian Personal Computer magazines. However, no individual articles are notable for use as a reference